Accurate Modeling and Efficient QoS Analysis of Adaptive Systems under Bursty Workload

Diego Perez-Palacin, Raffaela Mirandola, and José Merseguer

Abstract—Fulfillment of QoS requirements for services and applications deployed in the Internet is becoming a must. A widespread characteristic of systems that offer their services over the Internet is that they are usually subject to highly variable and bursty workloads. The allocation of as many resources as necessary to satisfy the applications' QoS requirement during peak workloads could entail a waste of computing resources. A solution for these situations is to provide the system with self-adaptive techniques that can allocate resources only when and where they are required for fitting such QoS. We pursue an accurate QoS evaluation of these workload-aware self-adaptive systems based on stochastic models. To this end, on this work we propose an accurate modeling of the workload variability and burstiness phenomena. We base our research on previous approaches which proposed the utilization of Markov Modulated Poisson Processes (MMPP). We extend these approaches in order to represent in more detail the parts of the workload that have more influence in the QoS offered by self-adaptive systems. Unfortunately, this stochastic modeling may lead to a non tractable QoS analysis of the system. This work also develops an efficient procedure for carrying out the QoS analysis of self-adaptive systems deployed in the Internet.

Index Terms—MMPP modeling; workload modeling; stochastic modeling; self-adaptive systems; Petri nets

1 Introduction

The Quality of Service (QoS) offered by a software application is an important matter for its successfulness in the marketplace. This is exacerbated nowadays, in the era of Internet services and online applications, where the popularity of software systems with good functionality may be jeopardized by poor QoS, such as low performance or scarce availability. Examples of site degradation due to high workloads of Internet traffic abound, from legitimate requests, such as "flash crowds" effects, to disruptions due to malicious requests, such as denial of service attacks.

In order to build software with good QoS, the formal methods community has achieved important advances [2]. Among other results, the QoS analysis leads to the identification of the amount of resources allowing a system to fulfill the required QoS (e.g.,[3], [22]). However, in the last years the deployment of software systems has changed. Currently, they are not constrained to execute using a predefined number of resources, on the contrary, they can dynamically adjust their deployment as a response to changes in their execution context, such as changes in the workload. This could be obtained, for example, exploiting the elasticity and auto-scaling properties offered by cloud-computing. Hence, software service providers can save costs during periods of time when the workload is low, but can also provide good QoS during workload peaks by provisioning an extra amount of resources temporarily. This type of systems are included in what it is called *self-adaptive software* [9].

- *Diego Perez-Palacin and Raffaela Mirandola, are with Politecnico di Milano, Italy*
- *José Merseguer is with Universidad de Zaragoza, Spain*

The model-based QoS analysis of a software that adapts its deployment to fulfill the required QoS while allocating the minimum amount of resources, is a challenging research topic that has not yet been completely addressed. Beyond traditional models of software behavior, we need models that represent, among others: adaptation policies, monitoring of the environment (to manage false positive adaptations or lacks of adaptations -false negatives-) and workload variations. In this work, we concentrate on the latter, the modeling of workload variations over time in the Internet, such as the requests supported by services, applications and websites.

It has been previously observed that the workload received by most of the systems, operating on the Internet, is highly variable and shows bursty behavior [6], [15], i.e., irregular spikes of congestion. Therefore, the models used to analyze QoS should be able to represent these characteristics. If the workload model does not account for the existing *burstiness*, then the model analysis can lead to optimistic results; e.g., it declares fair resource utilizations and probability of congestion, while in the real setting they would not be guaranteed. Formal methods that have proved to be useful in modeling workloads with *burstiness* in the arrival rate are the Markov Arrival Processes (MAP) and a concrete subtype of them, the Markov Modulated Poisson Process (MMPP) [11]. In particular, work on fitting MMPP and MAP parameters from workload traces with *burstiness* is very useful for the analysis of QoS properties, such as performance or availability, of a wide range of systems.

Considering workload-aware self-adaptive systems carefully, we can observe that their suitable configurations are different depending on the workload they are receiving. Hence these systems should adapt (e.g., provisioning or release of resources) when they recognise that the workload is changing. However, the usual techniques of MMPP fitting do not propose an accurate representation of the periods of time when the workload is changing. Although this fact does not prevent an accurate analysis of non-adaptive systems under variable workload, it hampers the precise analysis of self-adaptive systems due to the importance of such periods.

For example, a system model which does not take into account the periods of time in which the workload is changing will not be able to anticipate any workload change. The result will be a system that starts its adaptations when bursts of requests are already arriving. This can lead to too pessimistic performance and availability results from the model analysis.

In this work, we propose an accurate modeling and an efficient QoS analysis of self-adaptive systems that execute under variable and bursty workloads. The accurate modeling builds on our previous work in [21], where we were based on an MMPP(2) model, now we advance it to an MMPP(N) model, which allows for dealing with both, *short* and *long term* variability in the workload. However, this outstanding modeling when combined with the self-adaptive system model has a price, it may hamper the analysis, even it can turn into a non-tractable analysis problem. Hence, here we developed a procedure based on Markov reward models for carrying out the QoS analysis.

The rest of the paper is organized as follows. Section 2 exemplifies the need of precise models for representing bursty workloads of adaptive systems. Section 3 shows the usefulness of MMPPs for modeling bursty workloads, but also its limitations for self-adaptive systems. Sections 4, 5 and 6 address these limitations: regarding the accuracy in the modeling and the QoS analysis. Section 7 evaluates the feasibility of the solutions proposed in previous sections. Section 8 revises related work. Section 9 offers a conclusion.

2 Bursty Workload and Adaptive Systems

Goal of this section is twofold. We first show the usefulness of adaptive systems for fulfilling QoS requirements when subject to bursty workloads; then we demonstrate the need of advanced stochastic formalisms for modeling bursty workloads.

To this end, we use a real workload trace, illustrated in Figure 1, which plots the monitored arrival times of requests to the FIFA 1998 World Cup site [25]. The y axis counts the requests every ten seconds received by the Paris region server, while the x axis represents the flow of time. This trace, regardless its age, is yet one of the most detailed workload traces of a real service that are available on the Web for the public. The shape of the graph depicts a quite bursty workload: the mean arrival rate of requests is 46.9 per second, but during 90% of the time it is under 86.2 req/s, while there are many peaks of short duration whose arrival rate can easily reach 350 req/s (i.e., around 7 times higher than the mean).

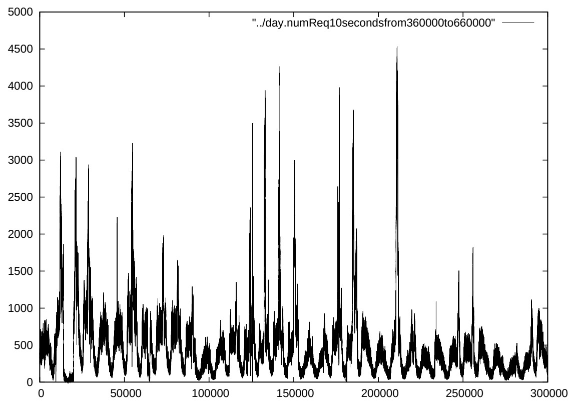


Fig. 1. Requests every 10 seconds

	No. servers used	Availability
Static system	7	99.07%
Adaptive system	3.62	99.943%

TABLE 1
Simulation results

Usefulness of an adaptive system versus a static one. Let us consider a very simple service composed of just a single computational activity requiring on average 25 milliseconds of processing time to serve a request. Since in the rest of the paper we use stochastic models with exponential distributions to model service times, we assume that these 25 milliseconds of processing time are exponentially distributed. In this way the model will represent perfectly the service time and we have eliminated a possible source of inaccuracies in the analysis, allowing us to concentrate on the precision of the workload model. In addition, the system can store in a queue up to ten requests, serving them following a FIFO policy. If a new request arrives when there are already 10 requests in the queue, it is discarded. The system availability requirement to satisfy is that *at least* 99% *of requests must be served.*

To study the characteristics of this system, we have implemented two Java programs: one that simulates a static system that uses c servers to process the requests in the queue, and another that simulates an adaptive system that dynamically modifies the number of used resources based on the amount of workload that is receiving in each moment. Table 1 shows the results we obtained from the execution of the Java programs using as workload input the data in the trace depicted in Figure 1.

For the static system, first row in Table 1 shows the experimental results: seven servers were required to achieve an availability of 99.07% (the availability using six servers was 98.31%).

For the adaptive system, we calculated for each number of servers the value of the arrival rate for which the service will show exactly the 99% of availability. Then, the system will decide to adapt to use one more server (i.e., from using c servers to $c + 1$) when the workload exceeds the calculated maximum arrival rate value for c servers. In turn, the system will decide to adapt to use one less server (i.e., from using $c + 1$ to c) when the workload is below the calculated maximum arrival rate value for $c - 2$ servers; i.e., we follow the well-known *hysteresis-based* approach to reduce the adaptations that are false positives. We consider that a server needs 1 minute for its booting time; i.e., the time period between the moment when the adaptation decision was launched to the first moment when the server is ready to server requests. Under these settings, second row in Table 1 shows the obtained results: 3.62 active servers were required in mean to achieve an availability of 99.94%. Another result of the adaptive system was that, when a server was switched on, it remained working for 9.34 hours in average. The number of servers used at any time is depicted in Figure 2.

As a conclusion we observe that adaptive systems offer better QoS results -more availability with less servers- than static ones. The former can actually accommodate the bursty periods by provisioning resources while releases them when they are no longer required.

Stochastic formalisms for modeling bursty workloads. Once it has been studied the system that receives the real bursty workload, we show the role of formalisms for its modeling. To this end, we performed a model-based analysis of the static system using a classical $M/M/c/B$ queuing model. The arrival process was a Poisson process with $\lambda = 46.9$ req/s, as in Figure 1. The service rate was exponentially distributed with mean $\mu = 40$, which is the inverse of the 25ms of the processing time required by our service. We evaluated the queue for a different number of servers c, being the system capacity $B = c + 10$, in order to represent the ten requests than can be enqueued. We obtained that using two servers the 99.91% of requests can be served, result that is pretty far away from the one obtained in the simulation that required seven servers for satisfying the 99.07% of requests.

The Poisson process as workload model is broadly used in stochastic analysis. However, it does not offer good results for modeling the bursty phenomenon, as we showed above. Being the goal of this work proposing an accurate modeling of bursty workloads for the QoS analysis of self-adaptive systems, we pursue more detailed descriptions of workload variability. Next section starts the discussion.

3 Bursty Workload Modeling for Adaptive Systems

Looking at a bursty workload trace we observe that the arrival rate is highly variable in the *long term*, meaning that the changes in the arrival rates differ by orders of magnitude. However, at the same time it is also highly

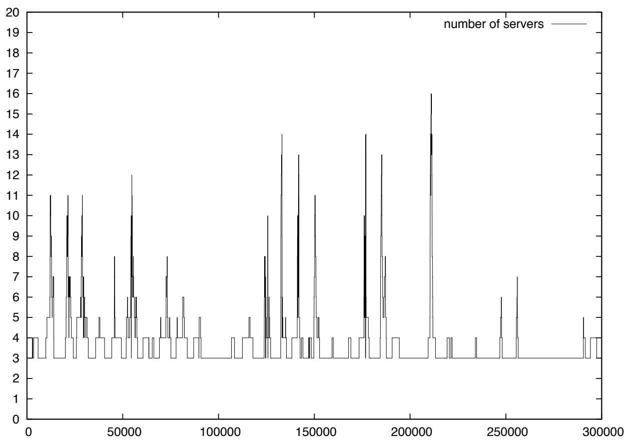


Fig. 2. Number of Active Servers

probable that in the *short term* (i.e., few seconds in the future) the arrival rate will not differ much from the current one. These long and short term views are well represented by the concept of *multiple workload states*, where each state is governed by an arrival rate. Since the arrival rate in each workload state is different, then, at a certain point in time it may be completely different from the arrival rate received a long time ago, hence, the long term view is well represented. Regarding the short term view, this is also modeled since the inter arrival time of requests at each state is not constant but they follow a probability distribution function. When the distribution is an exponential one, these concepts are exactly the ones represented by a Markov Modulated Poisson Process (MMPP).

3.1 Markov Modulated Poisson Processes and Workload Fitting

MMPPs have been largely and successfully used in the literature as workload models for systems evaluation, mainly network traffic [11], [12], [14]. MMPPs are suitable to model event arrival processes, high variability and auto-correlation for event generation. An MMPP is a stochastic process, the arrival rate at each moment is determined by the states of a continuous-time Markov chain (CTMC). So, when the chain is in state s_n, the arrival process is a Poisson process with rate λ_n. An MMPP with N states is defined by the NxN infinitesimal generator Q, which governs the state changes in the MMPP, and a vector Λ of N components representing the arrival rates in each state.

$$Q = \begin{pmatrix} -q_{11} & q_{12} & \cdots & q_{1N} \\ q_{21} & -q_{22} & \cdots & q_{2N} \\ & & \cdots & \\ q_{N1} & q_{N2} & \cdots & -q_{NN} \end{pmatrix}, \Lambda = (\lambda_1, ..., \lambda_N),$$

where $\forall n,\ \lambda_n > 0$ and $\forall n, n', (q_{nn'} \geq 0$ and $q_{nn} = \sum_{n':n' \neq n} q_{nn'})$.

To fit the parameters of the MMPP with N states we use the algorithm presented in [15] and surveyed in [17]. The input of this algorithm is a trace of counts of requests received in consecutive intervals of time. Let us denote with: $\mathcal{C}$ the trace of counts, $|\mathcal{C}|$ to amount of data in $\mathcal{C}$, and c_i the i-th value in $\mathcal{C}$ where $i \in [1...|\mathcal{C}|]$.

Algorithm in [15] first creates the vector of arrival rates Λ and then fits the transition rate values in Q. The process is briefly described in the following:

Create vector of arrival rates: To create the arrival rate in each state and, at the same time, to decide the value for the amount of states N, the algorithm uses the maximum and minimum values in $\mathcal{C}$, called $max(c_i)$ and $min(c_i)$ respectively.

- First, it assigns $\lambda_1 = (\sqrt{1 + max(c_i)} - 1)^2$.
- Then, to create every λ_n, $n > 1$, it iteratively applies the formula $\lambda_n = (\sqrt{\lambda_{n-1}} - a)^2$.
- The algorithm stops right after finding the first λ_n that satisfies $\lambda_n - a\sqrt{\lambda_n} \leq min(c_i)$.

Where the width parameter a has an arbitrary value (e.g., $a = 2$ in [15]). It represents the width, in number of standard deviations of the Poisson distribution, of the range of observations that will be associated with each arrival rate; e.g., $a = 2$ means that there will be associated to each arrival rate all the observations between its mean minus two standard deviations and its mean plus two standard deviations.

The value of n in the last iteration is assigned to N (i.e., the dimension of the MMPP) and the vector Λ is filled with the calculated λ_n values $1 \leq n \leq N$.

Fit the transition rate values: The fitting procedure of values in Q assumes that each c_i in the trace can only be produced by one state. Therefore, it exists the function $state(c_i)$ that, given a count of requests c_i ($i \in [1, |\mathcal{C}|]$), returns the state s_n that generated it ($1 \leq n \leq N$).

The behavior of function $state(c_i)$ is the following: $state(c_i) = s_n$ if and only if $((\lambda_n - a\sqrt{\lambda_n}) < c_i) \wedge (c_i \leq (\lambda_n + a\sqrt{\lambda_n}))$.

The univocal assignment of a state to a count of requests under this rule is possible because the algorithm followed to create λ_n values of the MMPP ensured that:

$$\forall_{n\in[1..N]}, \nexists\ n' \neq n \ \ | \ \ (\sqrt{\lambda_n} - a)^2 < \lambda_{n'} < (\sqrt{\lambda_n} + a)^2$$

Using this function, values $q_{nn',\ n\neq n'}$ in Q are calculated as the probability of being the count of requests in position $i+1$ produced by state $s_{n'}$ given that the count of requests in position i has been produced by state s_n (i.e., formally $P(state(c_{i+1}) = s_{n'} \mid state(c_i) = s_n)$).

Finally, values q_{nn} are calculated as $q_{nn} = \sum_{n':n'\neq n} q_{nn'}$.

Therefore, in order to obtain an accurate modeling of the bursty workload offering better results than those obtained using the Poisson process in the $M/M/c/B$ queue, we used the algorithm above for MMPP fitting of the workload trace in Figure 1.

Being $a = 2$, we got an MMPP of 34 states and we applied such MMPP(34) as workload model to the *static* system described in Section 2. In this way we obtain an $MMPP/M/c/B$ model whose analysis results show that $c = 7$ servers are necessary to obtain an availability of 99.11%. This is a very good result, pretty close to the 99.07% in Table 1. So the MMPP(34) largely improved the results obtained by the Poisson process in the $M/M/c/B$ queue. However, when the purpose is to evaluate a self-adaptive system instead of a static one, we have found limitations in MMPPs as workload models, described in the following section together with our proposals to deal with them.

3.2 MMPP(N) for Adaptive Systems

Problem: Model Accuracy Limitation

For the sake of problem illustration, consider the synthetic trace in Figure 3 and consider that the MMPP fitting algorithm above yields $N > 2$ states with arrival rates $\lambda_1 > \lambda_2 > ... > \lambda_N$. In this periodic trace, half of the times the arrival rate in the next state increases and half of times decreases. For instance, if we look at the six values where the arrival rate is 50, we can see that for three of them the next arrival rate is higher (when the workload is increasing) and for the other three the next arrival rate is lower (when the workload is decreasing). For this reason, using the fitting algorithm we will obtain for each intermediate state s_i, $1 < i < N$, that $p_{i,i+1} = p_{i,i-1} = 0.5$, which means that states s_{i-1} and s_{i+1} are equiprobable from s_i. This is statistically true indeed. However, the trace clearly shows that being in s_i and being the workload for instance increasing then the probability to change to s_{i+1} is zero since $\lambda_i > \lambda_{i+1}$.

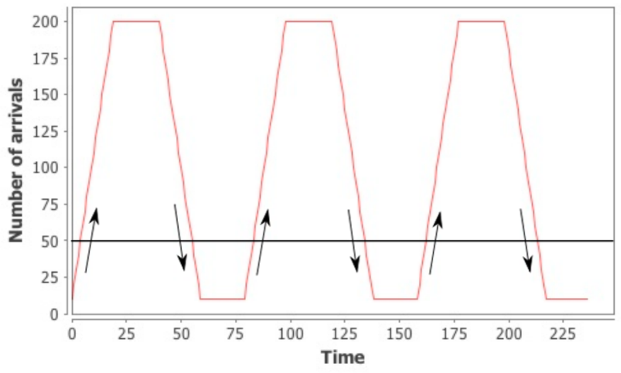


Fig. 3. A synthetic workload trace with constant increments and decrements

To illustrate this limitation, we fitted the parameters of an MMPP with the data in Figure 3. We obtained an MMPP(6). We simulated the arrival rate produced by the MMPP(6) in several experiments and we obtained the arrival rate changes depicted in Figure 4. Only the top-right chart brought a workload behavior similar to what it was expected. Other experiments did not produce the expected workload variations due to the following reasons:

- The workload started increasing soon after it had started to decrease, i.e., before reaching its minimum, as it happens in the top-middle and bottom-left chart.
- The workload starts decreasing before reaching its maximum, as it happens in the bottom-right chart.

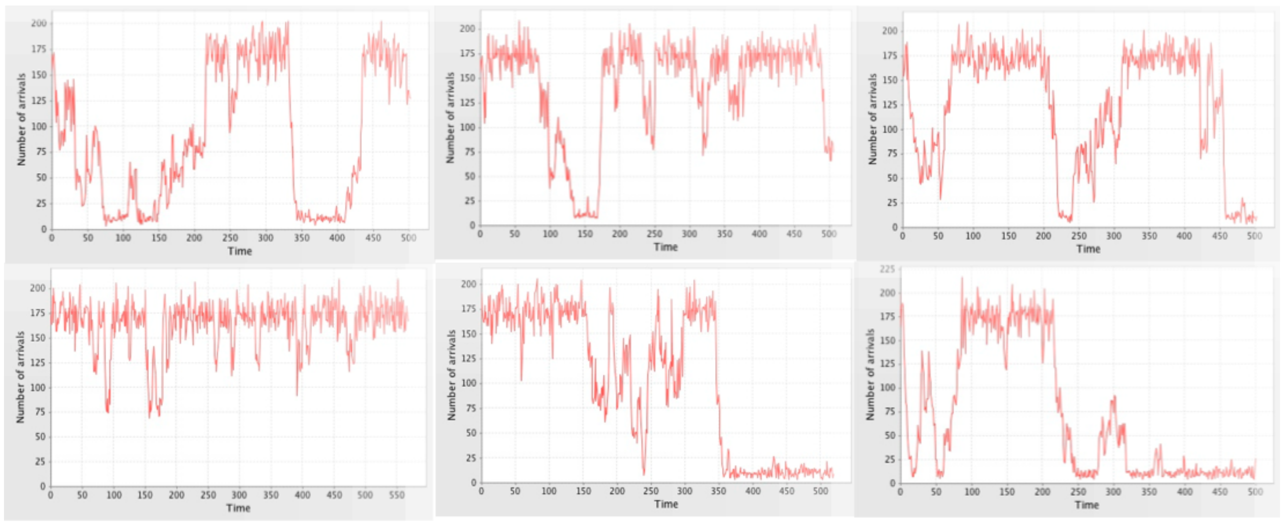


Fig. 4. Six samples of the arrival rate generated by the MMPP fitted using the data in Figure 3

- The workload, instead of ranging from the lowest to the highest, keeps its arrival rate too much time in a medium value, fact that happens in the bottom-middle chart.

Therefore, even though the fitting of the MMPP is correct from a statistical point of view, we can observe that this characteristic of the formalism is not very useful for modeling large continuous increments or decrements, say “tendencies”, in the workload. In fact this procedure for fitting MMPP(N) fails in the concept that the MMPP(2) used in [21] represented well, which was the the complete change between low workloads and bursts of requests.

Considering that we pursue the evaluation of self-adaptive systems that change their configuration depending on the workload they are receiving and that the work-load can show “tendencies”, then the statistically correct parameterization is not enough. Instead, we look forward to a workload model able to represent both bursts and “tendencies”.

Proposed solution

For a self-adaptive system that adapts in function of its workload, periods in which the workload is incrementing or decrementing are of special importance since they are the periods when the adaptation should be performed. Let us call them *transient periods*. For this reason, we propose to identify the *stable periods* in the workload (i.e., periods when the workload does not vary much), to keep only the states in the MMPP that represent these periods[1], and to model separately and explicitly the *transient periods*. Details on these steps are given in Sections 4 and 5. Section 4 details the state space reduction of the MMPP in order to keep only the most important states while Section 5 presents models for *transient periods*.

Problem: Model Analysis Limitation

1. From now on we will refer to *stable periods* as *stable states*, so to identify them with the MMPP states.

Let us consider the nominal behavior of a self-adaptive system stochastically modeled by $k1$ states, and its work-load modeled by $k2$ states. So, for system analysis, the aggregation produces a complete model that, in the worst case, has $k1 \cdot k2$ states. The reason is simple, for each system state the workload can be in any of its states. Accurate workload models, as the ones proposed in this work, may entail a large number of states, even considering the decision already taken about to reduce the number of states. Therefore, analysis techniques based on state-space enumeration, such as those that can be applied to Markov chains or Petri nets, should be carefully used in order to avoid state explosion, which would make the analysis intractable.

Proposed solution

To overcome this limitation, rather than analyzing the aggregated model, we propose to analyze the system availability for each *workload state* -stable or transient- separately, and to compose the results later. For doing this we apply Markov reward models (MRM) theory. Section 6 details our solution.

It is worth noting that for our solution to be useful, the inter-arrival time of requests should be much lower than the mean time spent in each *workload state* for each visit. Fortunately, this is what usually happens for Internet services, changes in the workload take several minutes or even hours, while inter-arrival times of requests are in the hundredths of second.

4 MMPP State Space Reduction

In this section we discuss methods that can reduce the amount of states yielded by the algorithm in [15]. To this end we could increase the value of a, however, the mean arrival rates of the states calculated in such way would only depend on the maximum and minimum arrival rate values in the trace $\mathcal{C}$. This fact shows some inconveniences since the workload states should be created by taking into account the characteristics of the arrival rate changes and not only its limit values. Therefore, we prefer to first create an MMPP using a small a, which will create a large amount of states, and then apply a filtering process to keep only those states in which the arrival rate is really most *stable*.

For reducing the number of states, we analyze different options discussed below: a) keep only the most visited states, b) keep only the states with longest mean sojourn time and c) keep only the states with longest maximum sojourn time.

For ease of explanation let us consider a workload-state trace $\mathcal{W}s$ of size $|\mathcal{C}|$, where each $\mathcal{W}s_i$, $i \in [1...N]$, indicates a workload state generated by c_i (i.e., $\mathcal{W}s_i = n \iff state(c_i) = s_n$).

a) Most visited states: For each state we count how many times it appears in $\mathcal{W}s$. We then could select as *stable* states those with most occurrences in $\mathcal{W}s$. Even if very simple and straightforward we did not adopt this method. The reason relies on the fact that the states with extreme associated arrival rates (e.g., the set of states with highest associated arrival rate) would not be considered because they usually have few occurrences. However, these states are very important, even if they rarely happen, since they represent worst-case scenarios. Overall this would be an insufficient model.

b) States with longest "mean sojourn time": First, for each state we count how many times it appears in $\mathcal{W}s$. Second, for each state we count how many times it happens that $\mathcal{W}s_i = n \ \wedge \mathcal{W}s_{i-1} \neq n$, which means how many times s_n has been visited from another state. Third, for each state we calculate its "mean sojourn time" (i.e., we divide the previous two values). Now we could select a percentage of states with the highest values. It is reasonable thinking that the states with large mean sojourn time are more *stable* than those states with low mean sojourn time. A reason can be that the latter show a low mean sojourn time because they are active only when the arrival rate is in the process of incrementing or decrementing. However this option shows also a drawback. The problem in this case is caused by the short-term variability of the workload. Consider for instance this small sub-trace of $\mathcal{W}s$ $[..., s_2, s_2, s_2, s_2, s_2, s_3, s_2, s_3, s_3, s_2, s_3, s_3, s_3, s_3, s_3, ...]$. In a long-term view, values in the beginning are continuously s_2 and values in the last part are continuously s_3, so both states should present long mean sojourn times. However, they will appear with short mean sojourn times since in a short-term view the trace presents many transitions from $s2$ to $s3$ and viceversa, which drastically reduces the result of their mean sojourn time. Therefore, the filtering of states obtained with this method is highly influenced by what is happening in the short-term arrival rate variability, which should not happen. For this reason this option is also considered not suitable to our needs. However, the main problem can be eluded by considering for each state its longest sojourn time only, as follows.

c) States with longest maximum "sojourn time": In this case we consider for each state its maximum number of consecutive occurrences in the trace. More formally, we could define an array $longestVisit$ of size N whose values are calculated as: $longestVisit_n = K$ such that

$$\exists_{i\in[1...|\mathcal{C}|]}\forall_{k\in[0...K-1]} \quad \mathcal{W}s_{i+k} = n$$
$$\wedge$$
$$\nexists_{i\in[1...|\mathcal{C}|]}\forall_{k\in[0...K]} \quad \mathcal{W}s_{i+k} = n$$

Consequently, each element in $longestVisit_n$ indicates the longest visit that s_n receives. We sum the values in the array as $sumL = \sum_n longestVisit_n$ and we keep the minimum set $\mathcal{S}$ of states whose sum of longest visits exceeds a given proportion p of $sumL$. Formally, we keep the set of states $\mathcal{S}$ such that

$$\sum_{s_n\in\mathcal{S}} longestVisit_n \geq sumL \cdot p$$
$$\wedge$$
$$\nexists \mathcal{S}' \quad ((|\mathcal{S}'| < |\mathcal{S}|) \wedge \sum_{s_n\in\mathcal{S}'} longestVisit_n \geq sumL \cdot p)$$

It is worth noting that, in this manner, the number of *stable* states is not preset by values p and N but it is tailored for each situation based on the relationships among values in $longestVisit$. This method reduces the influence of the short-term variability, but it is not perfect either. Consider for instance the sub-trace $[..., s_2, s_2, s_2, s_2, s_2, s_3, s_2, s_2, s_2, s_2, ...]$. In a long-term view, if s_2 appeared in the workload continuously, then the maximum sojourn time of $longestVisit_2$ would be at least ten. However, there is a change to s_3 in the sixth position, which makes $longestVisit_2$ to be at least five. This problem will not cause an error, because if a state is visited for long periods, in the long-term view, some of its visits will actually show a long visit period in the short-term -even if the maximum in each case does not completely hold the same value-. Therefore, if s_2 is usually visited in the long-term view for periods longer than five, even not finding any sub-trace with 10 consecutive values, there can be sub-traces with 9, 8 or 7 s_2 consecutive visits. Then, we can accept this weakness and consider this method suitable enough.

For these reasons, in our approach we adopt this method to obtain a new MMPP starting from the MMPP created by the algorithm in [15], as follows. Iteratively we remove from $longestVisit$ the state with lowest maximum "sojourn time". Assuming an iteration where s_n is removed then:

1) λ_n is removed then getting a new Λ.
2) $\mathcal{W}s$ is modified by overwriting to $\mathcal{W}s_i = \mathcal{W}s_i - 1$ all $\mathcal{W}s_i \geq n$ (or $\mathcal{W}s_i > n$ if $n = 1$).

Finally, the modified $\mathcal{W}s$ can be traversed for setting the transition rate values of the stochastic generator matrix Q.

Applying this reduction to our previous MMPP(34), that was created from the workload trace in Figure 1 using $a = 2$ and $p = 50\%$, we got an MMPP(12). The number of states has been reduced by 65%, meaning that 35% of the states accounted for more than 50% of the longest sojourn times.

5 Modeling Workload Transient Periods

This section describes the stochastic modeling of the workload *transient periods*, those that represent increments or decrements in the arrival rate. Consider that MMPPs model the transition from one state to another as an immediate event, however, not all workload traces show such abrupt behavior, they may have progressive increments and decrements. Consequently, the explicit modeling of these increments and decrements will improve the system analysis results. Moreover, since system adaptations should occur during these workload variations, an explicit modeling and analysis may provide us with valuable knowledge of the system behavior beyond the QoS.

The idea of this modeling is to represent large increments and decrements in the arrival rate while we keep a representation of the short-term variability. For example, Figure 5 shows both a) a long-term increment in the arrival rate from 500 to 3000 requests every 10 seconds in a period of time around 30 minutes, and b) a short-term variability where the arrival rate in $i+1$ is similar to the arrival rate in i but it may be higher, equal or even lower. This is the type of behavior that the MMPP modeling lacks.

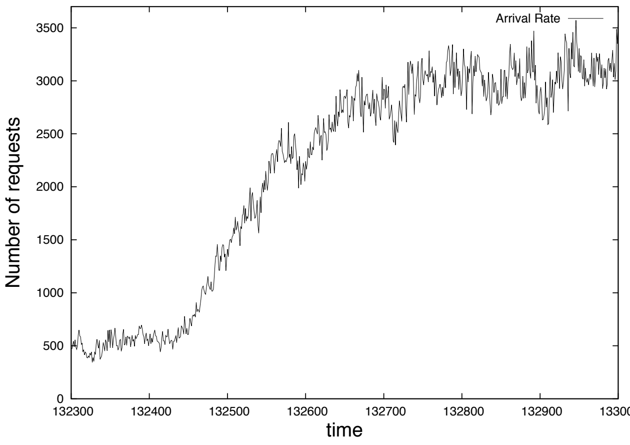


Fig. 5. Variation in the arrival rate. Zoom-in from indexes [132300...133000] in Figure 1

Consider Figure 6(a), it represents the MMPP model of a workload that changes from state s_n to $s_{n'}$ and viceversa. Although it represents long term variability, it clearly depicts an abrupt change. However, Figure 6(b) models *transient periods*. It represents long-term variability between arrival rates λ_n and $\lambda_{n'}$ as a linear increment -if $\lambda_n < \lambda_{n'}$- or as a linear decrement -if $\lambda_n > \lambda_{n'}$-. Note that, although this section deals with both short-term and long-term variability, for visibility reasons we have filtered out the short-term one in Figure 6.

For describing a linear increment or decrement we just need the mean time that the workload takes to change, $mt_{nn'}$ or $mt_{n'n}$. Note that the linear variation in the arrival rate can also be seen as the mean acceleration/deceleration of requests arrival, in $\frac{requests}{s^2}$.

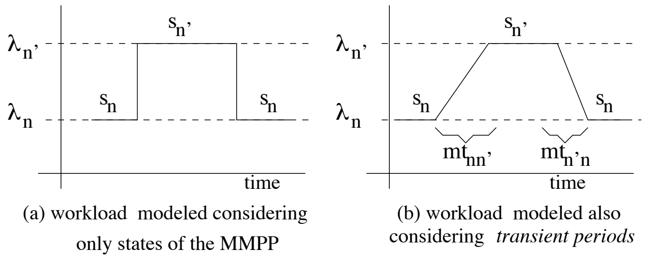

λn'
λn
sn'
sn
sn
time
(a) workload modeled considering only states of the MMPP
λn'
λn
sn'
sn
sn
mtnn'
mtn'n
time
(b) workload modeled also considering transient periods

Fig. 6. Workload models considering only states of the MMPP (a) and also including *transient periods* (b)

The models for the increments and decrements presented in this section will provide even more accurate results than the MMPP models presented so far while they still keep simplicity. Other approaches as curve fitting algorithms could offer more precise representations, however at the price of complexity.

5.1 Computation of Mean Changing Time between States

We devise two algorithms for calculating the mean time for the workload to move from a stable state s_n to another stable state $s_{n'}$; one for the case $\lambda_n < \lambda_{n'}$ and another for $\lambda_n > \lambda_{n'}$. These algorithms are based on the one presented in [21], which only dealt with the MMPP(2) case.

Let us explain in detail the algorithm for $\lambda_n < \lambda_{n'}$, i.e., a period of increment in the arrival rate; the other algorithm is very similar.

- **First step:** Data in $\mathcal{C}$ is traversed to find all intervals of increment from λ_n to $\lambda_{n'}$. Each interval is defined by its bounding positions in $\mathcal{C}$: $[init, end]$. Bound values *init* and *end* must satisfy that:

$$(c_{init-1} < \lambda_n) \;\wedge\; (c_{init} \geq \lambda_n) \;\wedge\; (c_{end+1} > \lambda_{n'})$$
$$\wedge$$
$$\nexists\, k \in [init, end] \;\;|\;\; c_k < \lambda_n \;\vee\; c_k > \lambda_{n'}$$

- **Second step:** Among all the intervals found, we discard those that do not contain a "real increment" in the workload. Due to the nature of the workload and the MMPP fitting algorithm, the workload may show different behaviors between λ_n and $\lambda_{n'}$, some of them could jeopardize the study. We want to discard intervals such as the one depicted in Figure 7(a), but keep intervals like the one in Figure 7(b). Hence, we decide to keep the intervals that show continuous increment in the workload and discard those that contain some arrival rate decrements. However, this distinction is not straightforward because values within the intervals also show short-term variability. Therefore, the short-term variability should be filtered out in order to decide whether an interval shows continuous increment. Algorithm below addresses this issue.
- **Third step.** For each interval selected by the previous step, calculate its length as $end-init$. Then, calculate

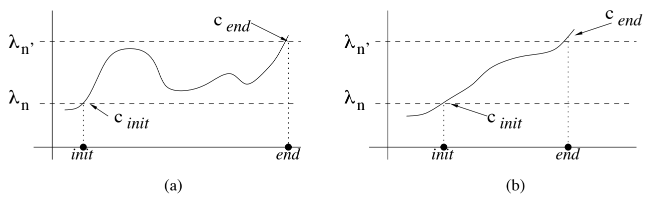


Fig. 7. Examples of workload increment

the mean of these lengths and assign such value to $mt_{nn'}$.

Algorithm for deciding "real increments":
Given an interval $[init, end]$, this algorithm decides whether it represents a "real increment" of the workload. The basis of the algorithm is to check whether the workload is continuously increasing in the long-term within the interval.

- *First step.* To filter out short-term variability, we add the number of requests received during L consecutive periods into a single value. So, the i-th aggregated value, where $i \in \{0, ..., \lfloor\frac{end-init}{L}\rfloor\}$, will be $\sum_{l=0}^{L-1} c_{init+iL+l}$.
L is a user's choice and represents how much we filter brief variations: a too low L would not filter short-term variability, while a too high L may recognize as continuous increments intervals that should be discarded.
- *Second step.* To check whether the workload is continuously increasing, we verify that

$$\forall_{i\in\{0,...,\lfloor\frac{end-init}{L}\rfloor-1\}}$$
$$\sum_{l=0}^{L-1} c_{init+iL+l} < \sum_{l=0}^{L-1} c_{init+(i+1)L+l}$$

5.2 Stochastic Models for Transient Periods

In order to analyze the system behavior during workload *transient periods* we need to model them, to this end we use GSPN [1], which is a language well suited for modeling the system behavior. We present two GSPN models, which use the previously computed $mt_{nn'}$: one for increments and the other for decrements.

GSPN Model for Workload Increment is represented in Figure 8(a). The arrival of requests is modeled by tokens in place $P_{arrivals}$. The idea is to augment the arrival rate from λ_n to $\lambda_{n'}$ in a mean of $mt_{nn'}$ time units.

The rate at which tokens are generated is $\lambda_n + \lambda_{inc} \cdot \#P_{inc}$.[2] In the beginning, $\#P_{inc} = 0$, then the arrival rate is λ_n, which is increased until $\lambda_{n'}$ along ω steps[3] . The parameters for achieving the objective of reaching $\lambda_{n'}$ in $mt_{nn'}$ time units are set as follows:

- $\lambda_{inc} = \frac{\lambda_{n'}-\lambda_n}{\omega}$, since we need ω steps to reach $\lambda_{n'}$.

2. $\#P_{inc}$ means the number of tokens in place P_{inc}.
3. It is worth noting that we are considering *infinite server* semantic for all transitions.

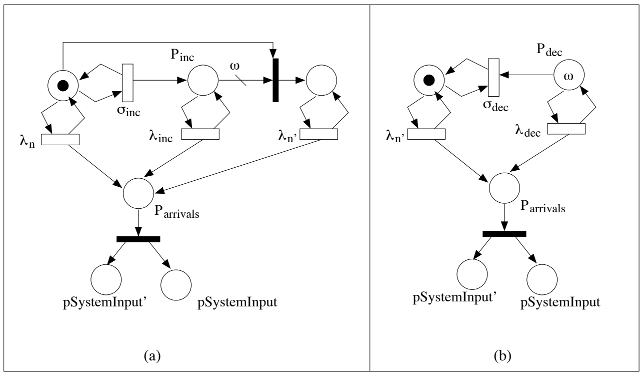


Fig. 8. Transient period GSPN models: (a) workload increment and (b) workload decrement

- σ_{inc} represents the rate at which increments in the arrival rate happen. Since ω increments occur in $mt_{nn'}$ time units, then $\sigma_{inc} = \frac{\omega}{mt_{nn'}}$.
- ω is a user's choice. The higher, the more accurate the modeling, however at the cost of increasing the state space. Figure 9 illustrates examples for $\omega = \{5, 10, 20, 50\}$.

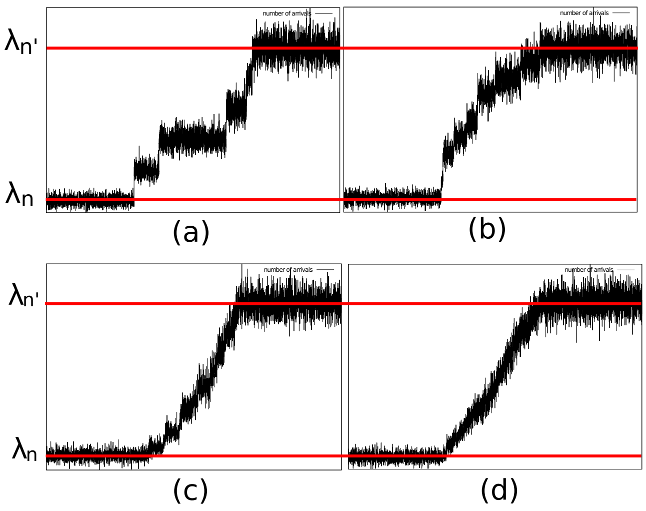


Fig. 9. Four examples of workload models using (a) $\omega = 5$, (b) $\omega = 10$, (c) $\omega = 20$, (d) $\omega = 50$

GSPN Model for Workload Decrement is represented in Figure 8(b). In this case tokens in $P_{arrivals}$ are created at rate $\lambda_{n'} + \lambda_{dec} \cdot \#P_{dec}$. In the beginning, $\#P_{dec} = \omega$, which decreases at rate σ_{dec}. The parameters are set as follows:

- $\lambda_{dec} = \frac{\lambda_n - \lambda_{n'}}{\omega}$.
- $\sigma_{dec} = \frac{\omega}{mt_{nn'}}$.
- ω is again a user's choice.

6 Efficient QoS Analysis Guided by Workload Models

Now we address the analysis of the stochastic models that we have proposed for an accurate workload modeling. This section develops a tractable QoS analysis, that aims at avoiding the potential state-space explosion that would happen if the whole model is analyzed at once. This proposal is based on Markov reward models (MRM) principles. The states in the MRM will be the union of the states in the MMPP created in Section 4 (called *stable workload* states) and a state for each of the needed transient models (called *transient workload* states).

Subsections 6.1 and 6.2 tackle the calculation of the reward for *stable workload* states and *transient workload* periods respectively. Then, Subsection 6.3 focusses on the generation of the CTMC that governs the MRM and the achievement of QoS results.

The solution here proposed advances the one in [21], where we *aggregated* the transient workload models with an MMPP(2) for stable states -one for bursty arrivals and one for non-bursty-. However, if we followed such approach then the high number of states created by the current accurate modeling would result in a non-tractable analysis.

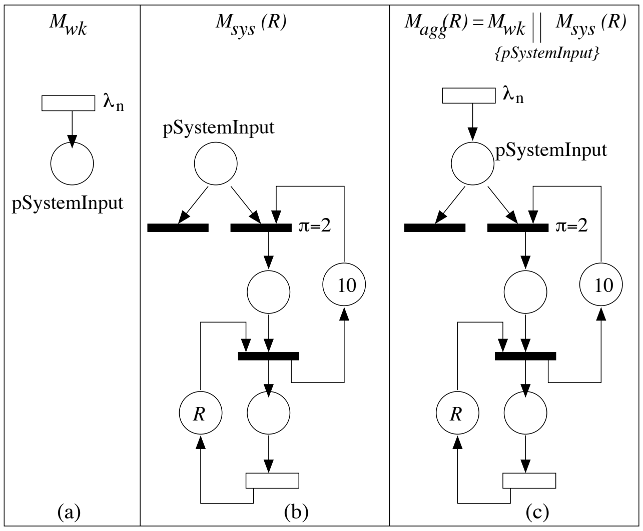


Fig. 10. Examples of GSPN models: (a) M_{wk}, (b) $M_{sys}(R)$ of the system in Section 2 and (c) $M_{agg}(R)$

6.1 QoS Analysis in Stable Workload States

In *stable workload* states the system only suffers short-term variability, that we model with exponentially distributed inter-arrival times -a standard technique for system QoS analysis-. Short-term variability should not be a reason to adapt the system since it happens with a frequency of seconds or few minutes: such a frequent adaptation rate would create a too unstable system. So, we assume that for each *stable workload* state an expected system configuration exists.

We create a parametric Markovian model called $M_{sys}(R)$ representing the system nominal behavior using *R* resources. As workload model we consider for each state s_n in the MMPP(N) proposed in Section 4, a Markovian model called M_{wk}, which represents an arrival rate of requests with λ_n. For system QoS analysis in s_n (i.e., whose results will become the reward values of the MRM for s_n) we create an aggregated[4] model $M_{agg}(R) = M_{sys}(R)||M_{wk}$. We can analyze $M_{agg}(R)$ to obtain: 1) the expected number of resources r_n under workload M_{wk}; and 2) results for the QoS metrics we pursue (e.g., response time).

Figure 10 depicts examples of these models in the GSPN language: part (a) a workload M_{wk}, part (b) the system illustrated in Section 2 $M_{sys}(R)$ and part (c) the aggregation of these models $M_{agg}(R)$.

6.2 QoS Analysis in Transient Workload Periods

In *transient workload* periods long-term variability is represented and consequently system adaptation for allocating resources occurs. These periods are placed between *stable workload* states, as represented in Figure 9. Let us call $s_{nn'}$ the transient period between stable states s_n and $s_{n'}$. We create:

- $M_{sys}(R)$ as in previous subsection.
- A Markovian model, M_{adapt}, that represents the adaptation logic. We follow a simple model for the adaptation logic since this concern is not the main scope the paper.
- A Markovian model, M_{wk}, that represents the increment (if $\lambda_n < \lambda_{n'}$) or decrement (if $\lambda_n > \lambda_{n'}$) in the workload, as described in Section 5.

For system QoS analysis in $s_{nn'}$ (i.e., the result that will be the reward of the MRM for $s_{nn'}$) we create an aggregated model $M_{agg} = M_{sys}(r_n)||M_{wk}||M_{adapt}$, where $M_{sys}(r_n)$ is an instance of $M_{sys}(R)$, $R = r_n$, r_n calculated as in previous subsection for stable state s_n. The main behavior of M_{agg} represents how the system serves requests while the workload is changing and the number of allocated resources vary from r_n to $r_{n'}$ according to the adaptation logic.

We can analyze M_{agg} to obtain: 1) results for the QoS metrics we pursue; and 2) the mean time between the moment when the arrival rate starts incrementing until the moment when both the arrival rate has reached $\lambda_{n'}$ and the system is in a configuration to serve requests at such rate. Using the latter and the $mt_{nn'}$, calculated in Section 5.1, we can obtain the mean time required by the system to finish its adaptation when the workload has already reached $\lambda_{n'}$, let us call it $mt_{nn'}^{adapt}$.

Figure 11 depicts examples of these models using GSPN. Part (a) models a workload increment, M_{wk}, according to Section 5.2. Requests are represented by tokens in places $pSystemInput$ and $pSystemInput'$, the first used by M_{adapt} and the latter by $M_{sys}(r_n)$. Part (b) models the system in Section 2 when using r_n resources. Part (c) models a simple rule-based adaptation logic. The logic adds resources as a function of the arrival rate during the last time interval of length $t_{Interval}^{-1}$. Concretely, if the number of requests is higher than x_1, then it adds up to res_1 resources; if it is higher than x_2, then up to res_2; and if

4. For aggregation we use theory of place composition proposed in [10].

it is higher than x_3, then up to res_3. Finally, M_{adapt} also models, through transition t_{setup}, the time required to boot resources -servers and application- before they are ready to serve requests.

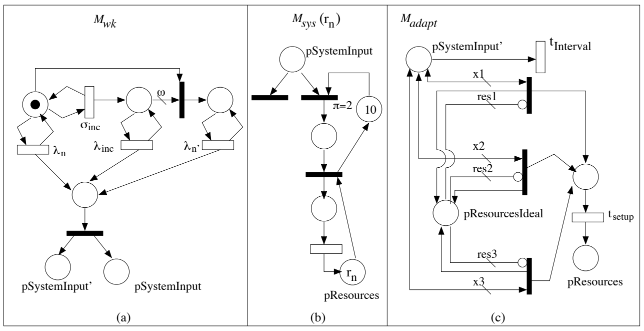


Fig. 11. Examples of GSPN models: (a) M_{wk}, (b) $M_{sys}(r_n)$ and (c) M_{adapt}

Figure 12 depicts the aggregation of the three models in Figure 11, composing them by places `pSystemInput`, `pSystemInput'` and `pResources`. This Petri net depicts a cycle of workload increment and adaptation. We decided to obtain the QoS results by using a steady-state analysis of the Petri net. So, we tuned the model to behave as a regenerative process of the cycle. We then added an immediate transition that fires when the cycle is finished for creating the initial marking again.

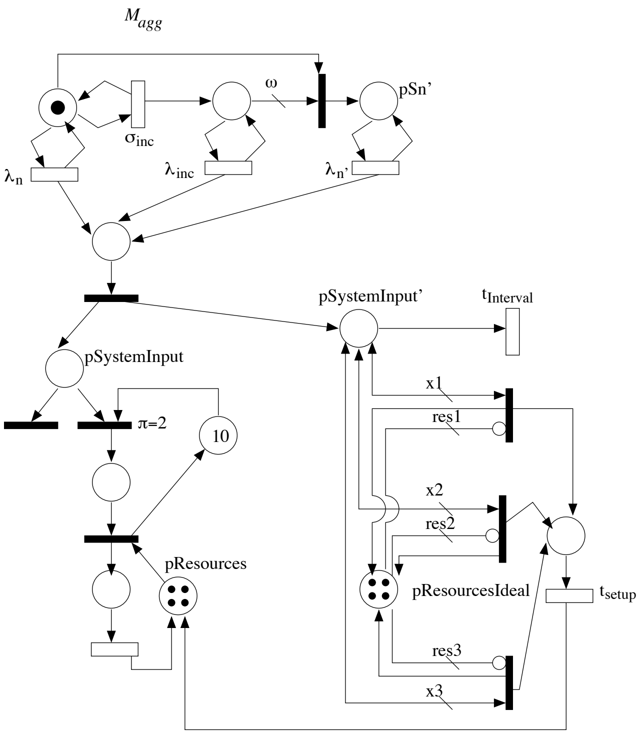


Fig. 12. GSPN aggregated model, where $r_n = 4$

6.3 Generation and Analysis of an MRM

As already stated, the system QoS analysis is accomplished through a Markov reward model (MRM). We initially consider that the states of the MRM are those of our MMPP(N); the state transition matrix of the MRM, Q^{MRM}, will initially be Q, the CTMC given by the infinitesimal generator of the MMPP. Then, we need to extend the MRM to also consider the *transient periods*.

For including a transient period $s_{nn'}$, we:

1) remove transition $Q^{MRM}_{nn'}$;

2) add a state $s_{nn'}$ in Q^{MRM};

3) add a transition from s_n to $s_{nn'}$ with rate $Q_{nn'}$ in Q^{MRM};

4) add a transition from $s_{nn'}$ to $s_{n'}$ with rate $\frac{1}{mt_{nn'}+mt^{adapt}_{nn'}}$ in Q^{MRM}. Therefore, $Q^{MRM}_{n\to n',n\to n'} = -Q^{MRM}_{n\to n',n'}$ and for each $j \neq n'$, $Q^{MRM}_{n\to n',j} = 0$.

Figure 13 graphically details the inclusion of three *transient states*[5] in a CTMC.

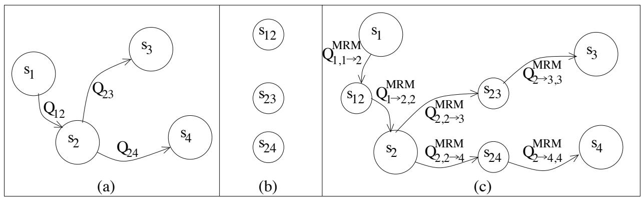
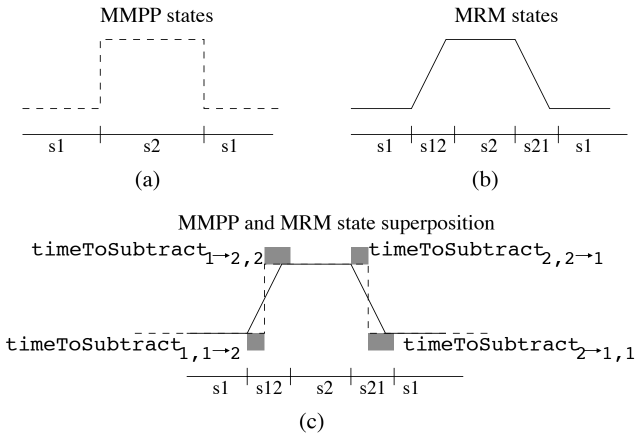


Fig. 13. Modification of the CTMC: (a) Q, (b) *transient* states, (c) Q^{MRM}

Q^{MRM} needs to be adjusted because currently the time spent in *transient states* is also represented by the *stable states*, so we have to appropriately reduce the mean time in *stable states*. Figure 14 illustrates an example of this issue. For a stable state s_n, we have to consider the time already included in transient states that s_n reaches and those reached by s_n, as follows:



MMPP states
MRM states
s1
s2
s12
s21
(a)
(b)
MMPP and MRM state superposition
timeToSubtract$_{1\to2,2}$
timeToSubtract$_{2,2\to1}$
timeToSubtract$_{1,1\to2}$
timeToSubtract$_{2\to1,1}$
(c)

Fig. 14. Time modeled in both states, stable and transient

5. Since we have converted a transient period into a state of the MRM, from now on we will call transient states to transient periods.

- For each transient state s_{xn} that reaches s_n we consider $timeToSubtract_{x \to n,n} = mt_{xn}/2 + mt_{xn}^{adapt}$. The rationale is that mt_{xn} (i.e., mean time incrementing or decrementing) should be represented only by the *transient* state but now it is also considered in the origin and target *stable* states. Then, we subtract half of it from the origin and half from the target (this case). On the other hand, mt_{xn}^{adapt} (i.e., mean time to adapt) is included in the *transient* state and in the target state s_n, hence we subtract it from the latter. For example, in Figure 14(c),
$$timeToSubtract_{1 \to 2,2} = \frac{mt_{12}}{2} + mt_{12}^{adapt}$$
For each s_n we consider $meanTTSinputs_n$ as the mean of all $timeToSubtract_{x \to n,n}$.
- For each transient state s_{nx} reached by s_n we consider $timeToSubtract_{n,n \to x} = mt_{nx}/2$. In this case we subtract the half part corresponding to the origen, as explained before. We do not decrement adaptation time here since such a time should only affect the target states. For example, in Figure 14(c),
$$timeToSubtract_{2,2 \to 1} = \frac{mt_{21}}{2}$$
For each s_n we consider $meanTTSoutputs_n$ as the mean of all $timeToSubtact_{n,n \to x}$.

Finally, the mean sojourn time in state s_n is
$$\frac{-1}{Q_{nn}} - meanTTSinputs_n - meanTTSoutputs_n$$
and we appropriately update Q^{MRM} as
$$Q_{nn}^{MRM} = \frac{-1}{\frac{-1}{Q_{nn}} - meanTTSinputs_n - meanTTSoutputs_n}$$
We also need to update in Q^{MRM} the state transition rates of each *stable* state s_n to ensure that the sum of all its rates is equal to $-Q_{nn}^{MRM}$, while preserving the transition probabilities. For doing this, all transition rates from s_n are scaled by factor $\frac{Q_{nn}^{MRM}}{Q_{nn}}$.

For example, in Figure 13(c):
$$Q_{22}^{MRM} = \frac{-1}{\frac{-1}{Q_{22}} - (\frac{mt_{12}}{2} + mt_{12}^{adapt}) - (\frac{\frac{mt_{23}}{2} + \frac{mt_{24}}{2}}{2})}$$
and
$$Q_{2,2 \to 3}^{MRM} = Q_{23} \frac{Q_{22}^{MRM}}{Q_{22}}$$

Now Q^{MRM} correctly characterizes the CTMC governing the MRM states -stable and transient-. We calculate the steady-state probability distribution of the CTMC, π^T, as the solution of $\pi^T Q^{MRM} = 0$ where $\pi^T \mathbf{1} = 1$. We refer to π_n as the probability of being in a stable state s_n and to $\pi_{nn'}$ as the probability of being in a transient state $s_{nn'}$. The steady-state expected reward, which is the steady-state QoS in this case, is calculated as usually [24], [13]:
$$qos = (\sum_{\forall\ s_n} \pi_n qos_n) + (\sum_{\forall\ s_{nn'}} \pi_{nn'} qos_{nn'})$$

a	num States	*a*	num States	*a*	num States
1	67	5	14	9	8
2	34	6	12	10	7
3	23	7	10		
4	17	8	9		

TABLE 2
States of the MMPP following the algorithm in [15] using different values for parameter *a*

Let us finally note that, although Q^{MRM} may potentially consist of N *stable* states plus $(N^2 - N)$ *transient* states, some *transient* states are not required for analysis. For example, between stable states s_n and $s_{n'}$ we do not need a transient one when:

- There is no one-step transition from s_n to $s_{n'}$, i.e., $q_{nn'} = 0$. Here we benefit from the reduction on the number of stable workload states and state transitions performed, since it helps to have a Q matrix populated with a high proportion of zeros.
- The expected system configuration in s_n is the same as in $s_{n'}$, i.e., $r_n = r_{n'}$, in this case there is no need to adapt the system during the workload change.

7 Evaluation

This section presents an evaluation of the proposed approach based on the simple software service described in Section 2, which is represented by the GSPN model depicted in Figure 10(b) named $M_{system}(R)$, working under the workload trace depicted in Figure 1.

In previous sections, it has been shown the behavior of some parts of the approach under different parameters. For example, the motivation for proposing a model that explicitly considers transient periods of workload rather than a pure MMPP model was described in Section 3 and graphically depicted in Figure 4.

Hereafter, we separate the evaluation into two parts. Since the proposed approach requires a set of parameters, in the first part of the evaluation we have performed a series of experiments for creating the workload model under different parameter values in order to gain insights of its behavior and evaluate its potential outcomes. The second part of the evaluation presents and discusses the quality of the provided results in terms of system availability evaluation.

7.1 Evaluation of outcome of the algorithms with different parameter values

This section presents the results of executing the approach for creating the workload model under different values of its parameters. In previous sections these parameters were named a, p, L and ω.

Parameter a: Regarding the width parameter a, described in Section 3.1 for calculating the initial number of states in the MMPP and the arrival rate in each state, we have

a=1 Initial states = 67								
p	**num States**	**%**	***p***	**num States**	**%**	***p***	**num States**	**%**
1	67	0%	0.7	39	42%	0.4	20	70%
0.9	55	18%	0.6	32	52%	0.3	14	79%
0.8	48	31%	0.5	26	61%	0.2	9	87%

a=2 Initial states = 34								
p	**num States**	**%**	***p***	**num States**	**%**	***p***	**num States**	**%**
1	34	0%	0.7	19	44%	0.4	10	71%
0.9	27	21%	0.6	16	53%	0.3	7	79%
0.8	23	32%	0.5	12	65%	0.2	4	88%

a=3 Initial states = 23								
p	**num States**	**%**	***p***	**num States**	**%**	***p***	**num States**	**%**
1	23	0%	0.7	11	52%	0.4	6	74%
0.9	18	22%	0.6	9	61%	0.3	4	83%
0.8	14	39%	0.5	7	70%	0.2	3	87%

a=4 Initial states = 17								
p	**num States**	**%**	***p***	**num States**	**%**	***p***	**num States**	**%**
1	17	0%	0.7	9	47%	0.4	4	76%
0.9	13	24%	0.6	7	59%	0.3	3	82%
0.8	11	35%	0.5	5	71%	0.2	2	88%

TABLE 3
States of the reduced MMPP for different values of width parameter *a* and proportion parameter *p*

a=2 p=0.7 allIntervals=48407					
L	**real changes**	**L**	**real changes**	**L**	**real changes**
No L	22967	1	44177	2	46324
0.5	39639	1.5	45602	2.5	46749

a=2 p=0.6 allIntervals=32444					
L	**real changes**	**L**	**real changes**	**L**	**real changes**
No L	15231	1	29337	2	30797
0.5	26350	1.5	30286	2.5	31112

a=2 p=0.5 allIntervals=12990					
L	**real changes**	**L**	**real changes**	**L**	**real changes**
No L	5620	1	11360	2	12071
0.5	10000	1.5	11830	2.5	12220

a=2 p=0.4 allIntervals=3276					
L	**real changes**	**L**	**real changes**	**L**	**real changes**
No L	1300	1	2541	2	2733
0.5	2220	1.5	2657	2.5	2788

TABLE 4
Number of state changes and number of them that were considered real increments/decrements

experimented with values from 1 to 10. Table 2 presents the results.

It can be seen how, for low values, this parameter has a strong influence in the resulting number of states, while for large values the number of states proposed does not vary much. We will choose low values of a in order to create a large number of states and then reduce them according to other characteristics of the workload trace beyond its maximum and minimum values.

Parameter p: Regarding the parameter p described in Section 4, which represent the percentage value for reducing the number of states of the initial MMPP following the technique *states with longest maximum "sojourn time"*, we have experimented with values from $p = 100\%$ to $p = 20\%$ in steps of 10%. We have calculated the amount of states of the reduced MMPP when the initial MMPP was calculated with values from $a = 1$ to $a = 4$. These results are represented in Table 3, together with the percentage of reduction in the number of states resulting from pruning the percentage $100 - p$ of lowest maximum sojourn times.

In that table it can be seen that the states sojourn times are not homogeneous. For example, in the 10% of difference between using *p=100%* and *p=90%*, the number of states is reduced around the 20%. It means that the sojourn time of several states is much under the mean. However, in the 10% of difference between using *p=30%* and *p=20%* the number of states is reduced only around the 8%. This fact justifies the observation that there are states whose visits take much less time than other states, and the workload produced by these states can be considered as transient.

Parameter L: Regarding the parameter *L* described in Section 5.1, which is used to decide whether the intervals of workload variations can be considered "real increments" or "real decrements", we have experimented with a set of values that range from 30 seconds to 2.5 minutes in steps of 30 seconds. That is, when an interval of change in the workload between states is discovered, to filter out the short term variability, we group the number of requests in periods of length *L*. We have also checked the amount of real increments/decrements obtained if parameter *L* would not be used. Table 4 depicts, in function of *L* values (represented in minutes), the number of state changes that existed in the trace and the number of them that were considered real increments or decrements to calculate the mean changing times between all states. The results of the experiment that did not consider parameter *L* are represented in the table as "No L". We have observed the variation of *L* in the number of states created by setting $a = 2$ and ranging p between 0.7 and 0.4 in steps of 0.1.

It can be seen that, the larger the value of *L* the more changing intervals are considered real increments/decrements. The reason is that the short-term variability in the arrival rate is filtered in a higher degree, which in turn also entails that there are considered as real increments/decrements some of the intervals that should have been filtered out. It is worth noting that, in some cases, if the *L* parameter is not used, it was not found any real increment/decrement between two states. For example, this fact happened 5 times for the cases of *p=0.5* and *p=0.4*.

Parameter ω: Regarding the value of parameter ω described in Section 5.2, which is used to model stochastically the transient increments and decrements in the workload, we refer to the study shown in Figure 9 to show the effect of its different values.

7.2 Quality of availability results

The goal of the evaluation of the quality of results is to compare our analysis results with the system simulation results depicted in Table 1.

In this experiment, we used the workload trace depicted in Figure 1 and the system described in Section 2 and we evaluated its availability. We parameterized the process using the following values:

- Width parameter of states in the initial MMPP $a = 2$. We have chosen this value because it is small enough as to generate a high quantity of states in the MMPP. At the same time, this value is also large enough for producing states with arrival rates with sufficient separation. The latter is useful in case that two neighbor states are both selected as *stable states* by the state reduction algorithm. If arrivals in each state will be generated following the exponential distribution - as it happens in MMPP- , the value a=2 allows the identification of the state that generated each arrival in the workload log with high confidence during the fitting process.
- Percentage value p for reducing the number of states of the initial MMPP following the technique *states with longest maximum "sojourn time"* is 50%. We have chosen this value because, for *a=2* in Table 3, it is the threshold value from which the reduction in the number of states is less than the increment of p value (i.e., above $p = 50\%$, each increment of 10% in p creates a reduction lower than 10% in the number of states).
- Parameter L used to decide "real increments" and "real decrements" (described in Section 5.1) is set to one minute. Since each count in the workload trace represents a time interval of ten seconds, then $L = 6$.
- Parameter ω is set to 10.

With this setting, the proposed approach created 12 stable states (as corresponding to the value in Table 3 for *a=2* and *p=50%*) and 19 transient workload states (of which 14 represented an increment in the workload and 5 a decrement) with a corresponding MRM of 31 states. The value of availability that was assumed as the real system availability was the one obtained from the system simulation with the actual data in the workload log, i.e., the value 99.94% showed in Table 1. From the MRM evaluation we obtained a system availability resul of 99.96%.

To gain more insight about the behavior of the system, using the results of the MRM steady state evaluation, we have also calculated that the workload is in *stable* states the 98.4% of time, while the remaining 1.6% is in *transient* states. Moreover, we have measured that, in stable states, the system reaches an availability of 99.99%, while in transient states the availability is 99.27%

The importance of considering both the transient workload times and the system adaptation times in their model-based representations is motivated by the following facts:

- *if the model does not represent adaptation times*, it means that the model represents a system that could instantaneously change the amount of used resources in any moment. In other words, this representation is equivalent to assume that the system is always using the correct amount of resources. A similar concept is what our MRM would represent if it would only take into account the results in its *stable* states. The availability evaluation of these states provided us the result of 99.99%. Therefore, if the model did not represent adaptation times, we would fall into a high over overestimation of the system availability.
- *if the model does not represent transient times*, we can obtain and underestimation of the availability. The reason is that the model represents a system that starts its adaptation to use the correct amount of resources when the workload has completely changed. For example, there would be modeled that the system starts adapting when the workload is already high, then resulting in periods where the availability obtained from the model is lower than the real one. We do not provide a quantity for this result because the straightforward model of this situation incurs in the state-space explosion. However, in [21] is described an example of this kind of underestimation that was possible to quantify because the workload model consisted of only two states and hence it was tractable.

We can see that our approach has been able to calculate a system availability of 99.96% when the pursued value was 99.94%; so we are incurring still in an overestimation of the availability, whose origin has to be studied in depth.

8 Related work

Workload modelling and analysis has been widely recognised as a critical part for the design and development of dependable software systems, see, for example, the work on capacity planning by Menasce et al. [19], or the work of Serazzi et al. [4], [18] considering different types of applications, just to cite a few. Besides, it has been observed that the workload, for some kind of systems, is far from being stable but it presents high variability and shows burstiness. Therefore, if the workload model does not account for the existing burstiness, then the model analysis can lead to optimistic results. Research on workload and network traffic considering the burstiness in the arrival rate used MAPs and MMPPs [11] and their results show an accurate modeling of the workload variability, see for example [15], [17]. In particular, work on fitting MMPP and MAP parameters from workload traces with burstiness is very useful for the analysis of QoS properties of a wide range of systems. The parameter fitting of Markovian models, such as MMPPs, starting from traffic traces is a promising research field where works [14], [16], [20], [23], [7], [8] propose techniques that can be of interest for the self-adaptive systems field. Some of these fitting works also deal with the modeling of burstiness characteristic.

However, to the best of our knowledge, this aspect has been neglected in the modelling and analysis of self-adaptive systems. However, we think that this topic is of great importance for systems deployed on the Internet since it will help to improve their behaviour and overall performance and quality. So, starting from the results in [21], we strove for an extension to deal with more than two stable states, i.e., to advance towards an MMPP(N) modeling. In fact, we knew that if we were able to get such fine grain modeling then the accuracy of the QoS results would be of great quality, which was the main critical issue in our initial work. To this end we exploited the results obtained in [12], [5] to choose the estimators of the workload trace and the algorithm proposed in [15], [17] to fit the MMPP(N). Then, we had to deal with the QoS analysis problems we have described in the paper, which completely reshaped the proposal in [21]. The proposed QoS analysis is able to guarantee that the system meets the requirements in any state -stable or transient-. This is a restriction harder than to meet the requirements considering only a *fully aggregated* model as we have initially done in [21].

9 Conclusion

Current generation of systems deployed on the Internet needs substantial improvement to adequately manage workload issues. Unfortunately, it is very common to hear everyday news about degradations, disruptions or even complete fall down of systems. The situation will aggravate when deployments in the cloud become first class citizens, which will happen soon. Most of these QoS problems arise due to an inappropriate system management of the workload, which in the Internet is bursty and highly variable. Self-adaptive techniques offer a solution for Internet-deployed systems to adequately manage workload issues. However, the very burstiness and variability make the adaptation process challenging.

In this work we have proposed a model-based evaluation of the QoS of self-adaptive systems deployed on the Internet. Our solution addresses the problems and challenges previously described. Then, we proposed an accurate modeling of the burstiness and variability phenomena that allows identifying the adequate moments for system adaptation. Moreover, we leveraged stochastic models to attain an efficient QoS analysis of the system, which implies the use of Markov Modulated Poisson Processes and Markov reward models.

There are several interesting directions stemming from this work to be investigated. Currently, the transient periods are modelled with linear increments or decrements in the arrival rates. Different behaviours, such as logarithmic or exponential, could be analysed and compared with the linear ones. Another possible line of research include the consideration of different analysis techniques and their comparison with the Petri Nets adopted in this work. We are also working on the implementation of our methodology on a real testbed, to assess its effectiveness through a more comprehensive set of real experiments.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley Series in Parallel Computing - Chichester, 1995.

[2] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[3] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.

[4] M. Calzarossa and G. Serazzi. Construction and use of multiclass workload models. *Perform. Eval.*, 19(4):341–352, 1994.

[5] G. Casale, N. Mi, L. Cherkasova, and E. Smirni. Dealing with burstiness in multi-tier applications: Models and their parameterization. *IEEE Transactions on Software Engineering*, 38(5):1040–1053, 2012.

[6] G. Casale, N. Mi, and E. Smirni. Model-driven system capacity planning under workload burstiness. *IEEE Transactions on Computers*, 59(1):66–80, 2010.

[7] G. Casale, E. Z. Zhang, and E. Smirni. Kpc-toolbox: Best recipes for automatic trace fitting using markovian arrival processes. *Perform. Eval.*, 67:873–896, September 2010.

[8] G. Casale, E. Z. Zhang, and E. Smirni. Trace data characterization and fitting for markov modeling. *Perform. Eval.*, 67(2):61–79, Feb. 2010.

[9] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors. *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, volume 5525 of *Lecture Notes in Computer Science*. Springer, 2009.

[10] S. Donatelli and G. Franceschinis. The PSR methodology: Integrating hardware and software models. In J. Billington and W. Reisig, editors, *Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 1996.

[11] W. Fischer and K. Meier-Hellstern. The Markov-modulated Poisson process (MMPP) cookbook. *Perform. Eval.*, 18:149–171, September 1993.

[12] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *Selected Areas in Communications, IEEE Journal on*, 9(2):203 –211, feb 1991.

[13] B. Haverkort and K. Trivedi. Specification techniques for markov reward models. *Discrete Event Dynamic Systems*, 3(2-3):219–247, 1993.

[14] H. Heffes and D. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *Selected Areas in Communications, IEEE Journal on*, 4(6):856 – 868, sep 1986.

[15] D. P. Heyman and D. Lucantoni. Modeling multiple IP traffic streams with rate limits. *IEEE/ACM Trans. Netw.*, 11(6):948–958, Dec. 2003.

[16] A. Horváth and M. Telek. Markovian modeling of real data traffic: Heuristic phase type and map fitting of heavy tailed and fractal like samples. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 267–282. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-45798-4_17.

[17] F. Mallor, P. Mateo, and J. Moler. A comparison between several adjustment models to simulated teletraffic data. *Journal of Statistical Planning and Inference*, 137(12):3939 – 3953, 2007. ¡ce:title¿5th St. Petersburg Workshop on Simulation, Part II¡/ce:title¿.

[18] P. Manzoni, P. Cremonesi, and G. Serazzi. Workload models of vbr video traffic and their use in resource allocation policies. *IEEE/ACM Trans. Netw.*, 7(3):387–397, 1999.

[19] D. A. Menasce and A. F. A. Virgilio. *Scaling for E Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[20] H. Okamura and T. Dohi. Faster maximum likelihood estimation algorithms for markovian arrival processes. In *Quantitative Evaluation of Systems, 2009. QEST '09. Sixth International Conference on the*, pages 73 –82, sept. 2009.

[21] D. Perez-Palacin, J. Merseguer, and R. Mirandola. Analysis of bursty workload-aware self-adaptive systems. In *Third joint WOSP/SIPEW international conference on Performance Engineering - ICPE*, pages 75–84, Boston, USA, 05/2012 2012. ACM, ACM.

[22] D. Perez-Palacin, R. Mirandola, and J. Merseguer. Qos and energy management with petri nets: A self-adaptive framework. *Journal of Systems and Software*, 85(12):2796–2811, 2012.

[23] T. Rydén. An em algorithm for estimation in markov-modulated poisson processes. *Comput. Stat. Data Anal.*, 21:431–447, April 1996.

[24] K. S. Trivedi, J. K. Muppala, S. P. Woolet, and B. R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14(34):197 – 215, 1992.

[25] World Cup 1998 Access logs. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`. 1998.